🎯 Generate all combinations of n pairs of valid (well-formed) parentheses.

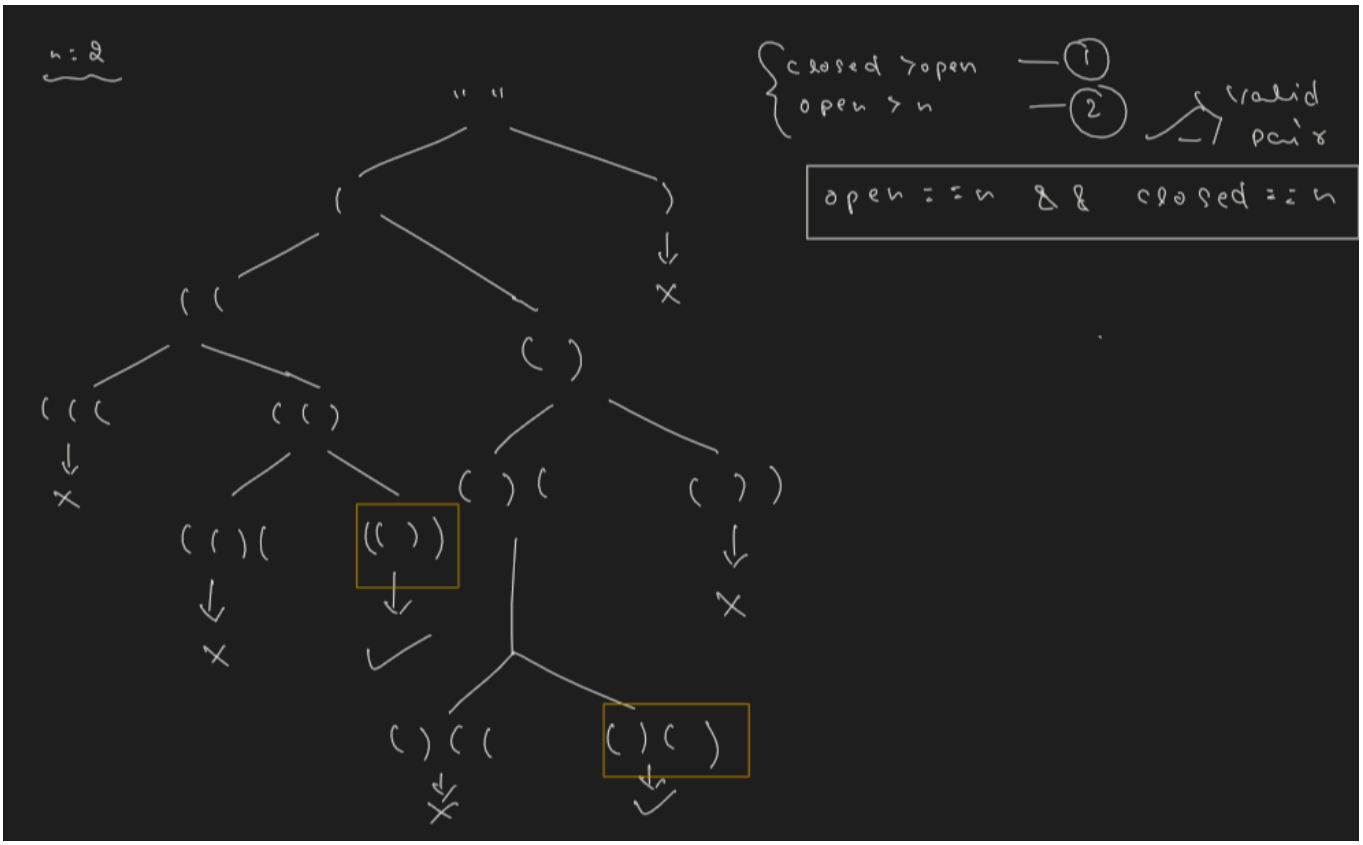
Given an integer n, return all possible strings that represent valid combinations of n pairs of parentheses.

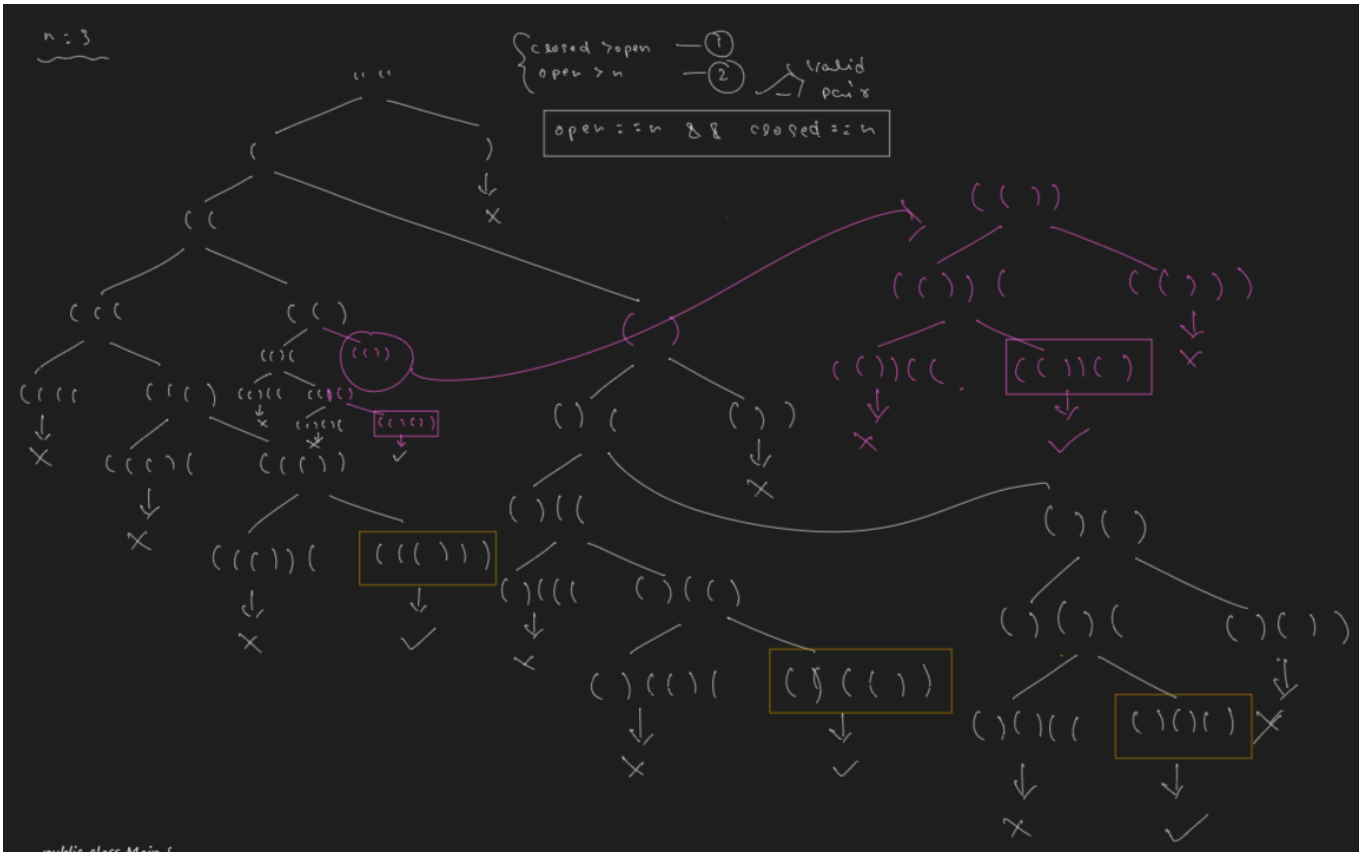
A valid parentheses combination must have:

- Each opening parenthesis `'('` properly matched with a closing parenthesis `')'`
- At no point in the string should the number of closing parentheses `')'` exceed the number of opening ones `'('`

```
n = 3;
["((()))", "(()())", "(())()", "()(())", "()()()"]
```

https://leetcode.com/problems/generate-parentheses/






```java
public class Main {
	public static void main(String[] args) {
		int n = 3;
		List<String> ll = new ArrayList<>();
		parentheses(n, 0, 0, "", ll);
		System.out.println(ll);
	}

	public static void parentheses(int n, int open, int closed, String ans,
List<String> ll) {
		if (open == n && closed == n) {
			ll.add(ans);
			return;
		}
		if (open > n || closed > open) {
			return;
		}
		parentheses(n, open + 1, closed, ans + "(", ll);
		parentheses(n, open, closed + 1, ans + ")", ll);
	}
}
```

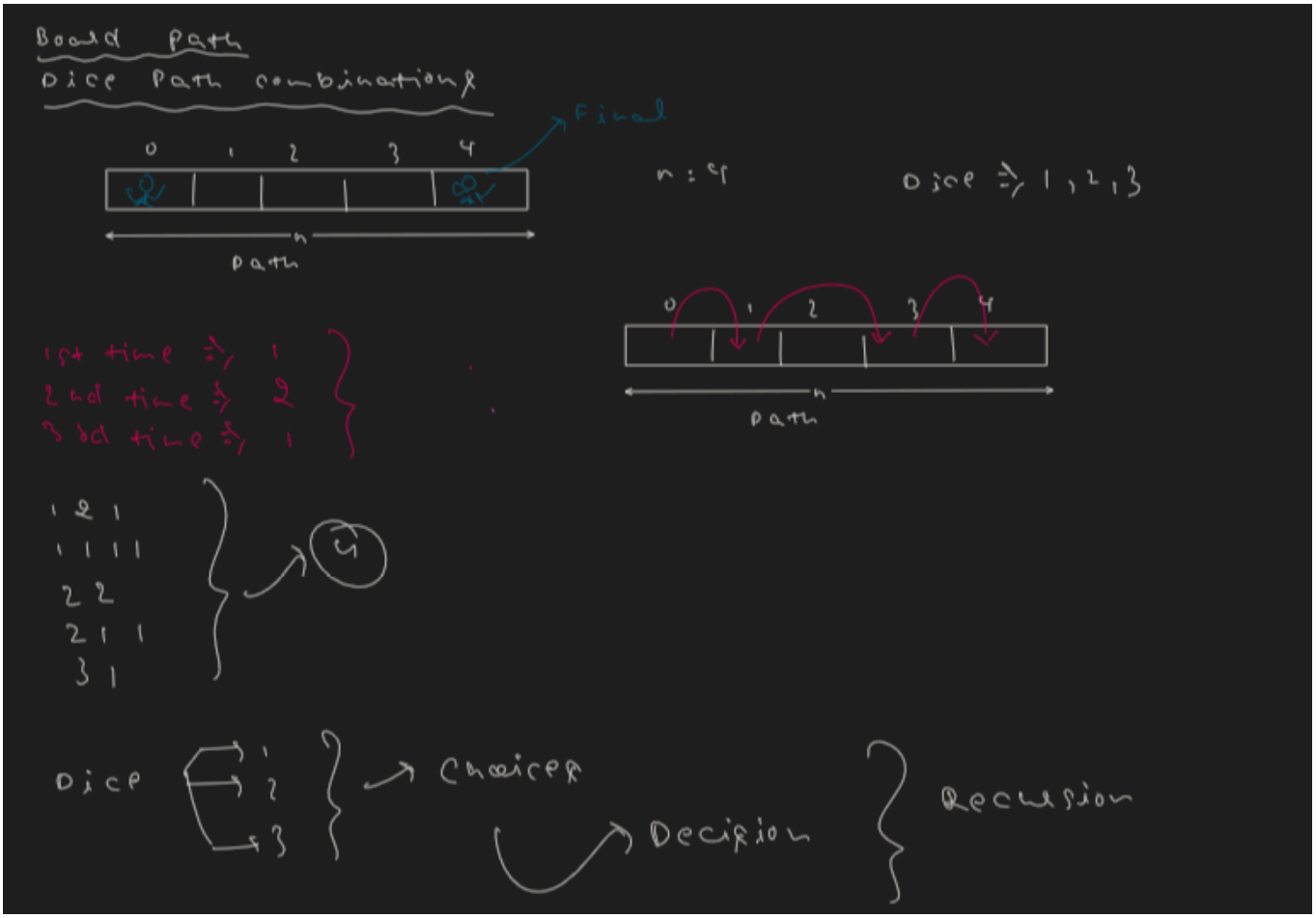
```java
class Solution {
    public List<String> generateParenthesis(int n) {
        return parentheses(n, 0, 0, "", new ArrayList<>());
    }

    public static List<String> parentheses(int n, int open, int closed, String ans,
List<String> ll) {
        if (open == n && closed == n) {
            ll.add(ans);
            return ll;
```

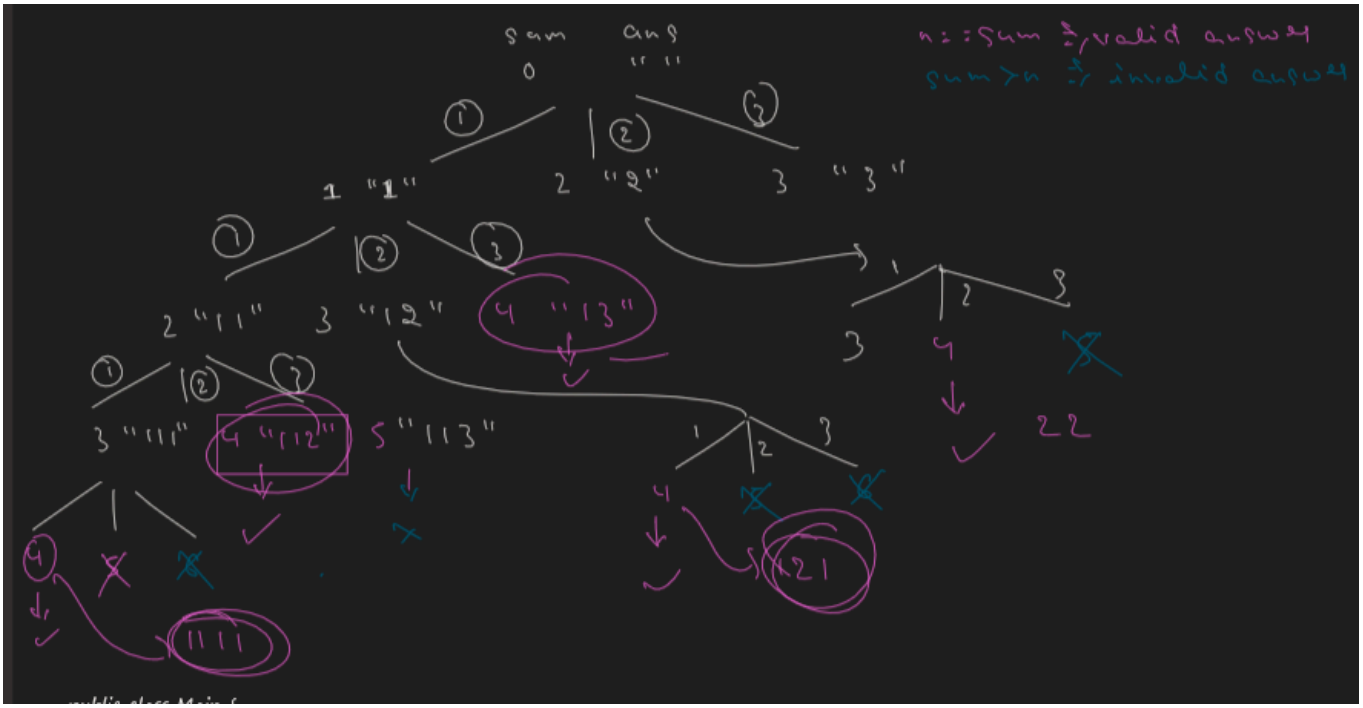
```
        }
        if (open > n || closed > open) {
            return ll;
        }
        parentheses(n, open + 1, closed, ans + "(", ll);
        parentheses(n, open, closed + 1, ans + ")", ll);
        return ll;
    }
}
```

🎯 Board Path/ Dice Path Combinations

You are at the starting point of a game board at position `0`, and your goal is to reach the end point at position `n`. You can move forward by rolling a dice, and in each move, the dice can result in a value of 1, 2, or 3 (i.e., you can move forward by 1, 2, or 3 steps at a time).

Write a program that prints all possible paths from position `0` to position `n`. Each path should be represented as a sequence of numbers where each number indicates the number of steps taken in that move.



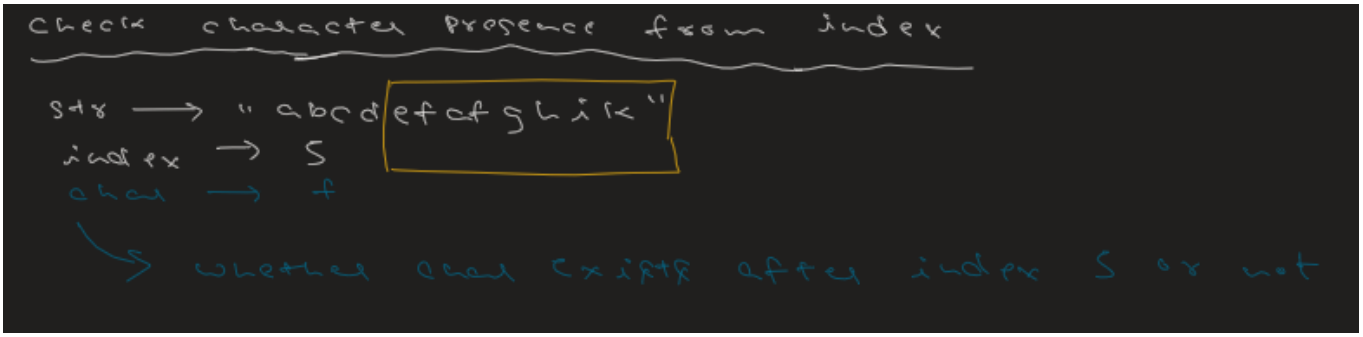



```java
public class Main {
	public static void main(String[] args) {
		int n = 4;
		printPath(n, 0, "");

	}

	public static void printPath(int n, int curr, String ans) {
		if (curr == n) {
			System.out.println(ans);
			return;
		}
		if (curr > n) {
			return;
		}
		// for (int dice = 1; dice <= 3; dice++) {
		//     printPath(n, curr + dice, ans + dice);
		// }
		printPath(n, curr + 1, ans + 1);
		printPath(n, curr + 2, ans + 2);
		printPath(n, curr + 3, ans + 3);
	}
}
```

🎯 Check Character Presence from Index

Given a string `str`, a character `ch`, and an integer `i`, write a method that returns `true` if the character `ch` is present in the string `str` at index `i` or any position after it. Otherwise, return `false`.

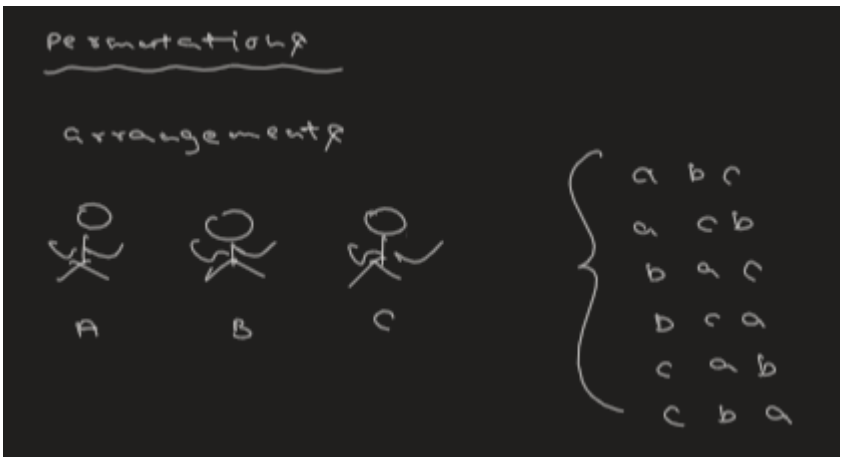
You must start checking from index `i`, not from the beginning of the string.

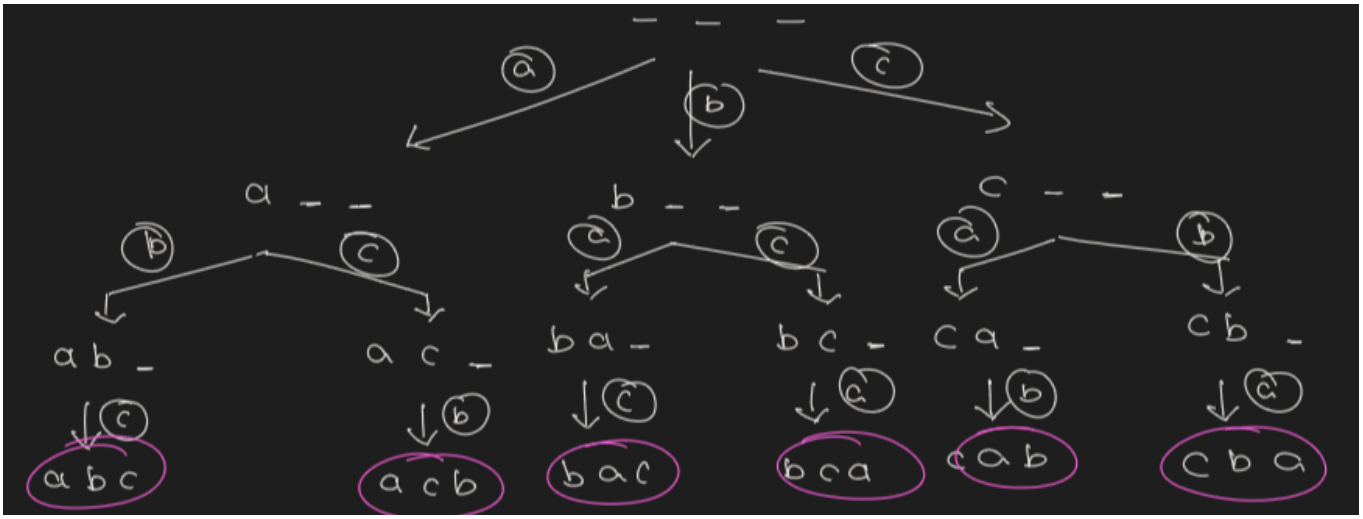


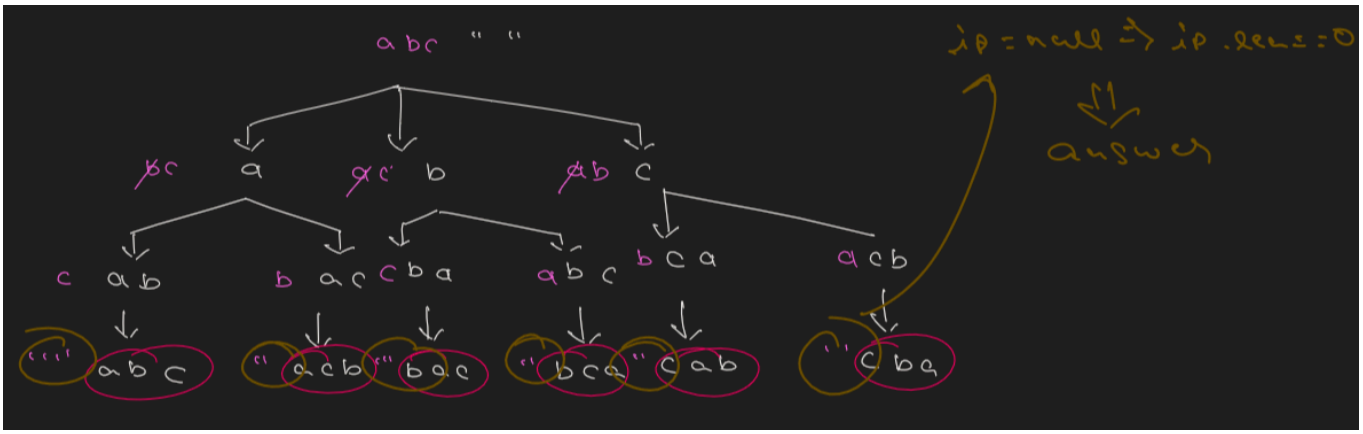

```java
public class Main {
	public static void main(String[] args) {
		String str = "ddfhag";
		int index = 2;
		char ch = 'h';
		System.out.println(isPresent(str, ch, index));
	}

	public static boolean isPresent(String str, char ch, int i) {
		for (int j = i; j < str.length(); j++) {
			if(str.charAt(j)==ch) {
				return true;
			}
		}
		return false;
	}
}
```

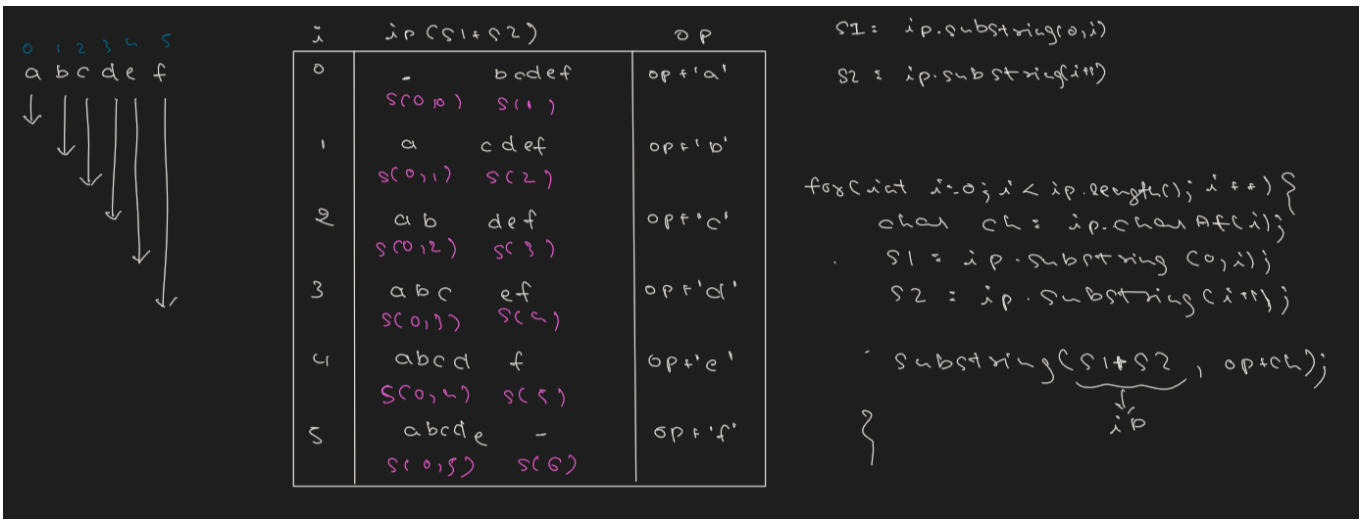
🎯 Print all permutations of a string

Permutation → arrangement
abc → abc, acb, bac, bca, cab, cba









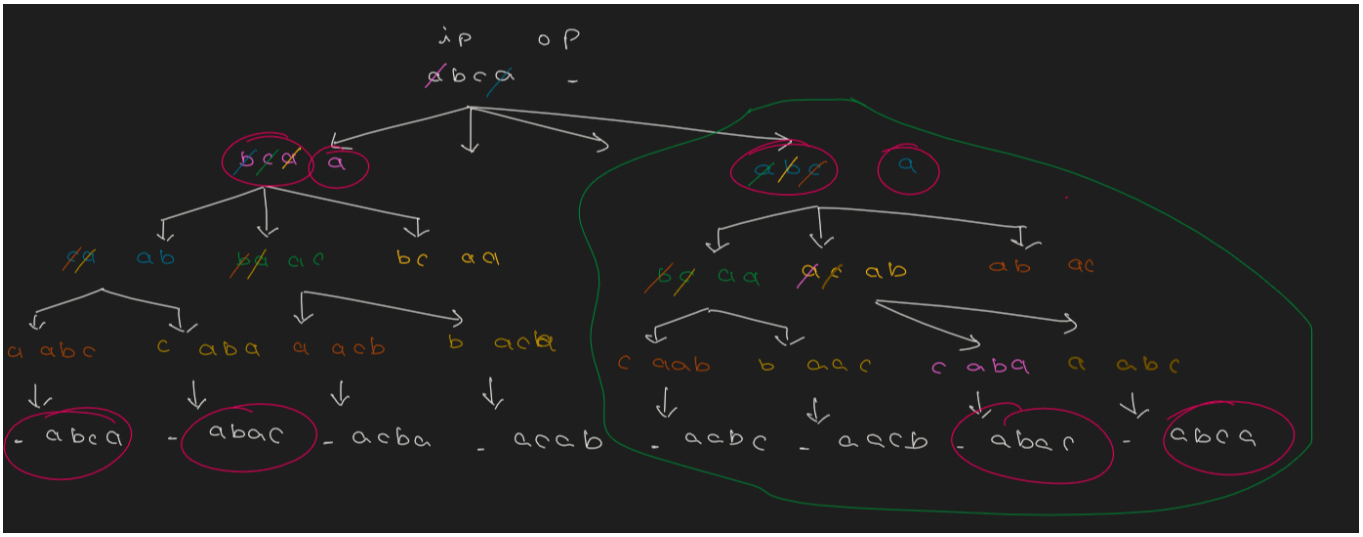

```java
public class Main {
	public static void main(String[] args) {
		String ip = "abc";
		String op = "";
		printPerm(ip, op);
	}

	public static void printPerm(String ip, String op) {
		if (ip.length() == 0) {
			System.out.println(op);
			return;
		}
		for (int i = 0; i < ip.length(); i++) {
			char ch = ip.charAt(i);
```

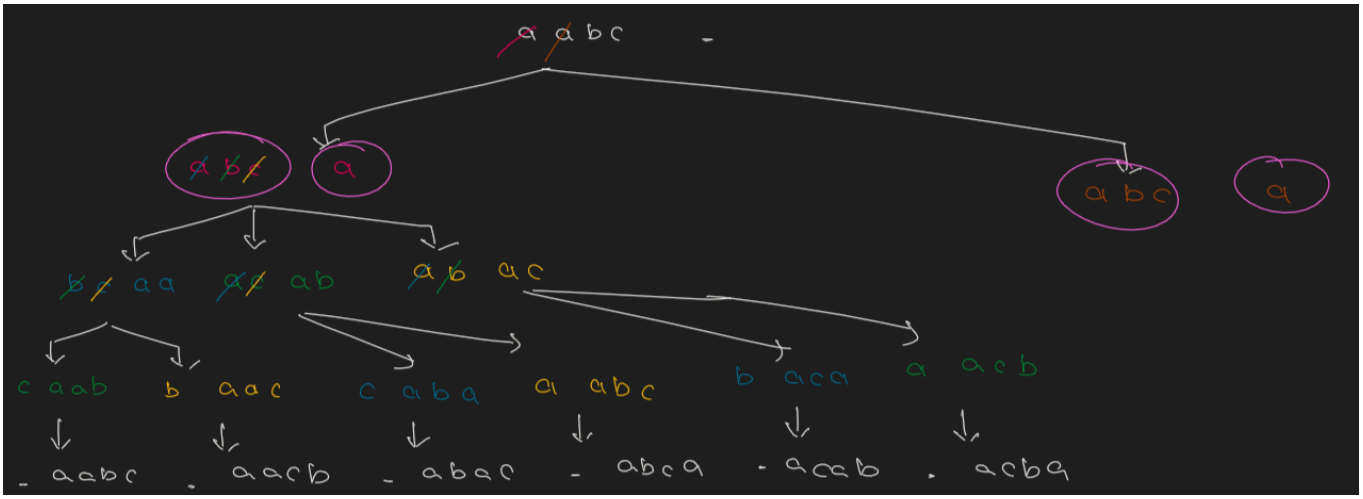
```
                String s1 = ip.substring(0, i);
                String s2 = ip.substring(i + 1);
                printPerm(s1 + s2, op + ch);
            }
        }
}
```

🎯 Print all unique permutations of a string

abca → abca, abac, acba, acab, aabc, aacb, baca, baac, bcaa, ~~bcaa~~, ~~baac~~, ~~baca~~, caba, caab, cbaa, ~~cbaa~~, ~~caab~~, ~~caba~~, ~~aabc~~, aacb, ~~abac~~, ~~abca~~, ~~acab~~, ~~acba~~

abca → bcaa, baac, baca, cbaa, caab, caba, aabc, aacb, abac, abca, acab, acba





```
public class Main {
        public static void main(String[] args) {
                String ip = "abca";
          String op = "";
                printPerm(ip, op);
        }

        public static void printPerm(String ip, String op) {
                if (ip.length() == 0) {
                        System.out.println(op);
                        return;
```

```
            }
            for (int i = 0; i < ip.length(); i++) {
                  char ch = ip.charAt(i);
                  if (isPresent(ip, ch, i + 1) == false) {
                        String s1 = ip.substring(0, i);
                        String s2 = ip.substring(i + 1);
                        printPerm(s1 + s2, op + ch);
                  }
            }
      }

      public static boolean isPresent(String str, char ch, int i) {
            for (int j = i; j < str.length(); j++) {
                  if(str.charAt(j)==ch) {
                        return true;
                  }
            }
            return false;
      }
}
```